(Rec)	TID	EventType	Category	Source	Application	Host	Severity
(1)	1001	TCPConnectionClose	Network	IO	System	Vesuvio	Low
(2)	1001	CiscoDCDLinkUp	Network	DHCP	Routing	Etna	Low
(3)	1001	AuditFailure	Security	Software	Authorization	Magna	High
(4)	1001	CoreDump	Memory	Exception	Kernel 2.4	Stromboli	High
(5)	1001	IRQConflict	Device	PCI Bus	System	Vulcano	High

FIG. 1

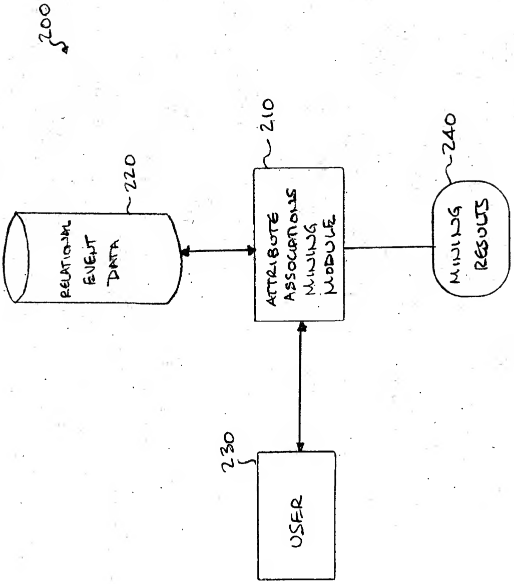
200
RELATIONAL EVENT DATA
220
ATTRIBUTE ASSOCIATIONS MINING MODULE
210
MINING RESULTS
240
USER
230

FIG 2

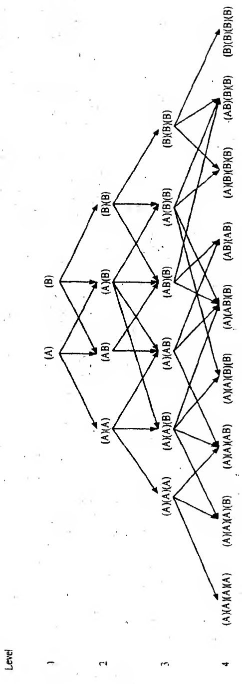
Level
1
2
3
4
(A)
(B)
(A)(A)
(AB)
(A)(B)
(B)(B)
(A)(A)(A)
(A)(A)(B)
(A)(AB)
(AB)(B)
(A)(B)(B)
(B)(B)(B)
(A)(A)(A)(A)
(A)(A)(A)(B)
(A)(A)(AB)
(A)(A)(B)(B)
(A)(AB)(B)
(AB)(AB)
(A)(B)(B)(B)
(AB)(B)(B)
(B)(B)(B)(B)

FIG. 3

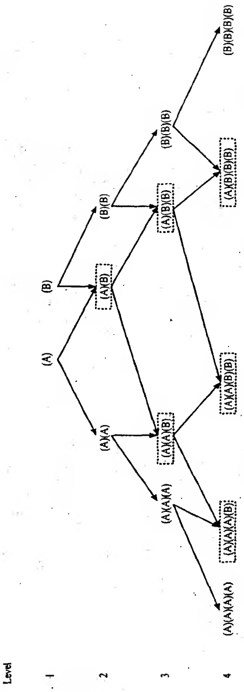
Level
1
2
3
4
(A)
(B)
(A)(A)
(A)(B)
(B)(B)
(A)(A)(A)
(A)(A)(B)
(A)(B)(B)
(B)(B)(B)
(A)(A)(A)(A)
(A)(A)(A)(B)
(A)(A)(B)(B)
(A)(B)(B)(B)
(B)(B)(B)(B)

FIG 4

$\mathcal{T}_1$	$\mathcal{T}_2$	
$t_{11} : A$	$t_{21} : B$	$t_{22} : C$

(a) $c = (A)(BC)$

$\mathcal{T}_1$	$\mathcal{T}_2$
$t_{11} : A$	$t_{21} : B$

(b) $c_3 = (A)(B)$

$\mathcal{T}_1$	$\mathcal{T}_2$
$t_{11} : A$	$t_{21} : C$

(c) $c_2 = (A)(C)$

$\mathcal{T}_1$	
$t_{11} : B$	$t_{12} : C$

(d) $c_1 = (BC)$

FIG. 5

$c = c_L \bowtie c_{L-1}$	c_L	c_{L-1}	Merge-Join?
(A)(A)(A)	(A)(A)	(A)(A)	Yes
(A)(A)(B)	(A)(A)	(A)(B)	Yes
(A)(AB)	(A)(A)	(A)(B)	No
(AB)(B)	(AB)	(A)(B)	Yes
(A)(B)(B)	(A)(B)	(A)(B)	Yes
(B)(B)(B)	(B)(B)	(B)(B)	Yes

FIG. 6

```
Methodology 1 HIFI(AttributeSet: $A$, Dataset: $D$, MinSupport: $min\_sup$)
generate frequent patterns for templates on the 1st level;
$L \leftarrow 2$;
$Template_L \leftarrow$ pair up templates on the 1st level to generate templates on the 2nd level;
$Cand_L \leftarrow$ join patterns on the 1st level to generate candidate patterns on the 2nd level;
while $Cand_L \neq \emptyset$ do
    countSupport($D, Cand_L$);
    eliminate candidates whose support are lower than $min\_sup$;
    $L \leftarrow L + 1$;
    $Template_L \leftarrow TemplateGen(Template_{L-1}, A)$;
    CandidateGen($Template_L$);
end while
return $\{t.patterns | t \in Template_L\}$;
```

FIG. 7

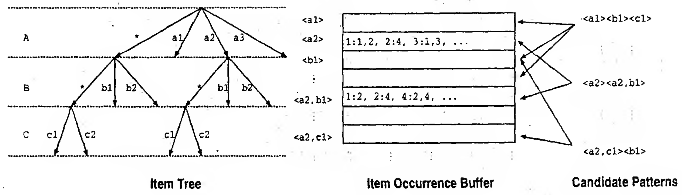
A
B
C
*
a1
a2
a3
*
b1
b2
*
b1
b2
c1
c2
c1
c2
<a1>
<a2>
<b1>
<a2, b1>
<a2, c1>
1:1,2, 2:4, 3:1,3, ...
1:2, 2:4, 4:2,4, ...
<a1><b1><c1>
<a2><a2, b1>
<a2, c1><b1>
Item Tree
Item Occurrence Buffer
Candidate Patterns

FIG. 8

Methodology 2 countSupport(Dataset: D, PatternSet: *candidates*)

```
for each tuple $t \in D$ do
  traverse the item tree to find all items supported by $t$;
  for each item supported by $t$ do
    record the current TID and RID in item's occurrence buffer;
  end for
  if any occurrence buffer is full then
    for each pattern $p \in candidates$ do
      scan the occurrence buffer of each item in $p$ (scan is synchronized by TID), and increase the count
      of $p$ if each item of $p$ is supported by a different record (of the same TID);
    end for
    empty all occurrence buffers;
  end if
end for
```

FIG. 9

```
Methodology 3 TemplateGen(SetOfTemplates: $parentTemplates$)
$Templates \leftarrow \emptyset$;
for each $p \in parentTemplates$ do
    for each child template $c$ of $p$ do
        $Templates \leftarrow Templates \cup \{c\}$ if all of $c$'s parent templates exist and have non-empty pattern set;
        prune $Templates$ using user preferences;
    end for
end for
return $Templates$;
```

FIG. 10

Methodology 4 CandidateGen(SetOfTemplates: *Templates*)

```
for each $c \in Templates$ do
  Let $c_i$ and $c_j$ be the two parents of $c$ that have the fewest number of patterns;
  if joining $c_L$ and $c_{L-1}$ is less costly than joining $c_i$ and $c_j$ then
    $c.patterns \leftarrow \text{mergeJoin}(c_L, c_{L-1})$;
  else
    sort the patterns in $c_i$ and $c_j$ by their common attributes;
    $c.patterns \leftarrow \text{mergeJoin}(c_i, c_j)$;
    sort the patterns in $c.patterns$;
  end if
  prune c.patterns if $c$ has value exclusion contraints;
  for each parent $c_k$ of $c$, and $c_k$ does not take part in the join operation do
    for each pattern $p \in c.patterns$ do
      remove $p$ from $c.patterns$ if the sub-pattern of $p$ with regard to $c_k$ does not exists in $c_k.patterns$;
    end for
  end for
end for
```

FIG. 11

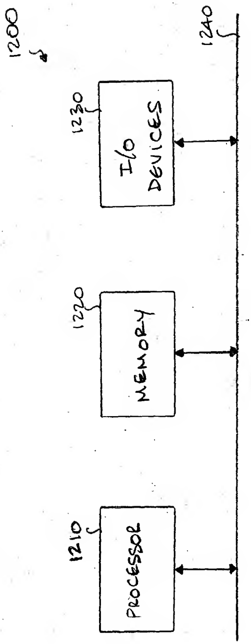

FIG. 12